# Keycloak instellen als credential-uitgever met OpenID4VCI

16 januari 2026 door Rodrick Awambeng, Forkim Enjeckayang, Ingrid Kamga, Bertrand Ogen

Voordat je Keycloak configureert, is het nuttig om de rol ervan in gedecentraliseerde identiteitsecosystemen te begrijpen. Als verifieerbare credential-uitgever kan Keycloak digitaal ondertekende credentials uitgeven met het OpenID for Verifiable Credential Issuesuance (OpenID4VCI)-protocol, waardoor vertrouwde partijen (ook wel verificeerders genoemd) deze onafhankelijk kunnen verifiëren zonder contact op te nemen met de uitgever.

Keycloak implementeert OpenID4VCI, waarmee verifieerbare inloggegevens (VC's) kunnen worden uitgegeven als digitale bewijs van identiteit of attributen. Het configureren van deze functionaliteit vereist consistente setup over de realm, clients en uitgegeven credentials (client scopes).

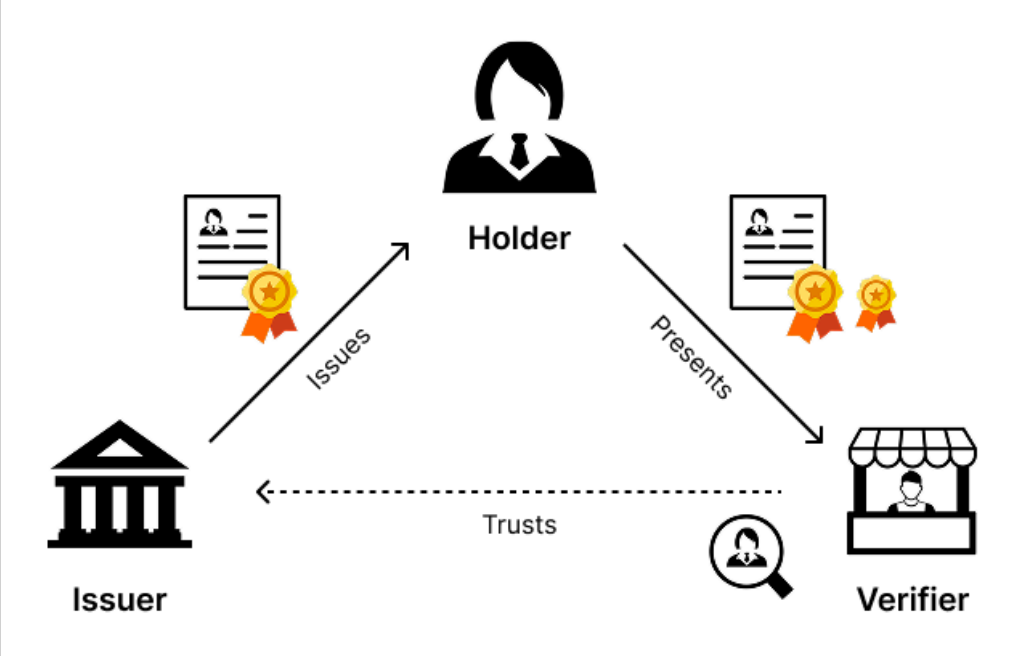
Beschouw bijvoorbeeld het volgende scenario: Het Keycloak OAuth SIG-team wil verifieerbare lidmaatschapsgegevens aan zijn leden verstrekken, inclusief hun naam en e-mailadres, die later kunnen worden getoond op evenementen ter plaatse of virtueel als bewijs van actief lidmaatschap.

Op het moment van schrijven van deze blog is de ondersteuning van Keycloak voor OpenID4VCI nog experimenteel. Deze gids gebruikt de **Keycloak 26.5.0 release**. De functie is in actieve ontwikkeling en wordt verwacht in de toekomst te worden gepromoot naar preview.

# Introductie tot OpenID4VCI & OpenID4VP

OpenID4VCI is een protocol ontwikkeld door de OpenID Foundation dat het OAuth 2.0-framework uitbreidt om de veilige en interoperabele uitgifte van verifieerbare inloggegevens (VC's) te ondersteunen. VC's zijn digitale, manipulatiebestendige weergaven van informatie, zoals identiteitsattributen of kwalificaties, die cryptografisch kunnen worden geverifieerd zonder dat je bij verificatie contact hoeft op te nemen met de uitgever.

Hoewel deze gids zich voornamelijk richt op **OpenID voor Verifiable Credential Issueance (OpenID4VCI),** omvat de algehele trustarchitectuur ook **OpenID for Verifiable Presentations (OpenID4VP),** die regelt hoe houders hun inloggegevens aan verificeerders presenteren. Samen maken deze protocollen gedecentraliseerde identiteitsecosystemen mogelijk, waarin gebruikers (houders) de controle over hun gegevens behouden en deze selectief kunnen delen met verificeerders.




**Figuur 1. Driehoek van vertrouwen of Emittent-Houder-Verificatiemodel**

In dit model:

- **Interacties tussen de houder van de uitgevende ↔** instelling worden geregeld door **OpenID4VCI,** dat de uitgifte van credentials dekt.

- **Interacties met Holder ↔ Verifiers** worden geregeld door **OpenID4VP,** dat de presentatie en verificatie van de credentials omvat.

Vanuit privacyperspectief is deze scheiding van zorgen fundamenteel. Door de **uitgifte van inloggegevens** los te koppelen van de **presentatie van de credentials**, kunnen uitgevers niet bijhouden waar, wanneer of hoe een houder zijn inloggegevens gebruikt. Deze architectuur voorkomt correlatie en profilering en zorgt ervoor dat gebruikers controle houden over hoe hun data wordt gedeeld.

# Waarom OpenID4VCI gebruiken

De belangrijkste motivaties voor het adopteren van OpenID4VCI zijn:

- **Interoperabiliteit**: Bouwt voort op gevestigde OpenID Connect (OIDC) standaarden, waardoor integratie met bestaande identiteitsproviders wordt vereenvoudigd.
- **Privacy en Beveiliging**: Ondersteunt selectieve openbaarmaking (bijvoorbeeld het aantonen van leeftijd zonder de geboortedatum te onthullen) en offline verificatie.
- **Compliance**: In lijn met regelgeving zoals eIDAS 2.0.
- **Efficiëntie**: Maakt gebruik van OIDC-mechanismen om de uitgifte te stroomlijnen terwijl het vertrouwen in het "driehoek van vertrouwen"-model (Issuer-Holder-Verifier) behouden blijft.

# Verifieerbare Credentialformaten

OpenID4VCI ondersteunt meerdere inlogformaten, die bepalen hoe verifieerbare inloggegevens worden gecodeerd en uitgegeven.

- SD-JWT VC – Selectieve openbaarmaking JSON Web Token verifieerbare credential
- JWT VC – JSON Web Token Verificeerbare Credential
- mDL/mdoc – Mobiel rijbewijs / mobiel documentformaat

Deze formaten stellen uitgevers in staat om draagbare, verifieerbare digitale bewijzen te leveren die selectieve openbaarmaking en offline verificatie ondersteunen in gedecentraliseerde identiteitsecosystemen.

**Keycloak ondersteunt momenteel zowel SD-JWT VC als JWT VC voor uitgifte**. mDL/mdoc, dat deel uitmaakt van de OpenID4VCI-specificatie, wordt verwacht in de toekomst ondersteund te worden.

# Concrete gebruiksscenario's mogelijk gemaakt door OpenID4VCI

Verifieerbare credentials openen een verscheidenheid aan toepassingen in de praktijk. Voorbeelden zijn:

- Overheden of niet-gouvernementele organisaties geven digitale identiteitskaarten of rijbewijzen uit die burgers kunnen tonen bij het boeken van hotels, het openen van bankrekeningen of het gebruik van openbare diensten, terwijl ze alleen de informatie die nodig is voor de transactie (bijv. leeftijd of woonplaats) onthullen zonder volledige persoonlijke gegevens te onthullen.
- Gemeenteraden geven verifieerbare geboorteaktes uit die universiteiten en ziekenhuizen kunnen valideren zonder een centrale zoekopdracht.
- Universiteiten geven digitale diploma's uit die werkgevers direct kunnen verifiëren op authenticiteit.
- Bedrijven geven werknemersbadges uit als verifieerbare inloggegevens voor kantoortoegang of externe authenticatie.
- Evenementorganisatoren geven verifieerbare tickets uit die offline kunnen worden gevalideerd.
- Professionele verenigingen geven lidmaatschapscertificaten uit, zoals in ons OAuth SIG-voorbeeld, om toegang tot afgesloten faciliteiten of conferentielocaties te verifiëren.
- Digitale film- of evenemententickets die manipuleerbaar zijn en offline kunnen worden geverifieerd.

Deze voorbeelden illustreren de verscheidenheid aan situaties waarin verifieerbare inloggegevens de noodzaak van directe communicatie tussen verificateur en uitgever wegnemen, terwijl het vertrouwen behouden blijft.

# Belangrijke stromen in OpenID4VCI

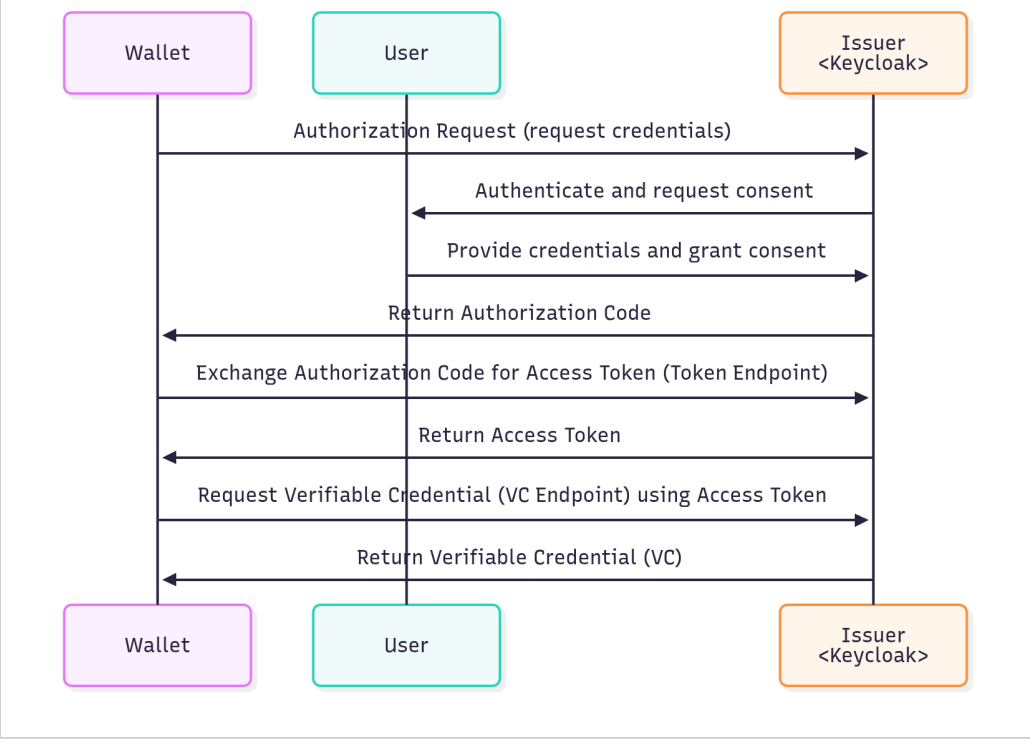
OpenID4VCI definieert twee primaire stromen voor het uitgeven van inloggegevens: de **Autorisatiecodestroom** en de **Pre-Authorized Code Flow**. Deze stromen bepalen hoe een wallet (de applicatie van de houder) een toegangstoken verkrijgt om een VC aan te vragen bij de uitgever.

## Autorisatiecode-flow

De Authorization Code Flow is **interactief** en vereist dat de houder authenticeert en toestemming geeft op het autorisatiepunt van de uitgever. Het is ideaal voor situaties waarin expliciete goedkeuring van de gebruiker vereist is, of waarbij aanvullende claims moeten worden geïnd.

**Stappen:**

1. De **Wallet** stuurt een autorisatieverzoek naar **het autorisatie-eindpunt van de uitgever**, waarin inloggegevens worden opgevraagd.
2. De **uitgever** authenticeert de **gebruiker** en vraagt toestemming voor de uitgifte.
3. De **gebruiker** verstrekt inloggegevens en geeft toestemming.
4. De **uitgever** stuurt een **autorisatiecode** terug naar de wallet.
5. De **Wallet** wisselt de code uit op het **Issuer Token Endpoint** voor een **Access Token**.
6. De **uitgever** retourneert de **Access Token** aan de wallet.
7. De **Wallet** vraagt het **verifieerbare inloggegevens** op bij het **Issuer Credential Endpoint** met behulp van het Access Token.
8. De **uitgever** retourneert de **VC** aan de wallet.



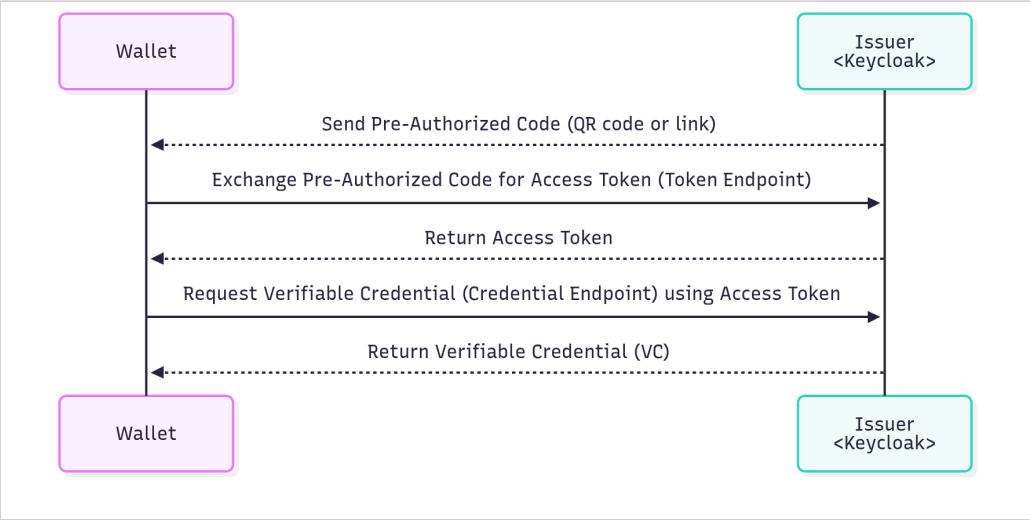

**Figuur 2. Credential-uitgifte via Authorization Code Flow**

## Vooraf-geautoriseerde codeflow

De Pre-Authorized Code Flow is **niet-interactief**. De uitgever authenticeert en autoriseert de gebruiker pre-authenticeert en verstrekt een **vooraf geautoriseerde code** (vaak via QR-code). Het is sneller en geschikt voor het uitgeven van vooraf goedgekeurde certificaten.

**Stappen:**

1. De **uitgever** verstrekt een **vooraf geautoriseerde code** aan de **wallet** (bijvoorbeeld via een QR-code of link).
2. De **Wallet** wisselt de code uit op het **Issuer Token Endpoint** voor een **Access Token**.
3. De **uitgever** retourneert de **Access Token** aan de wallet.
4. De **Wallet** vraagt het **verifieerbare inloggegevens** op bij het **Issuer Credential Endpoint** met behulp van het Access Token.
5. De **uitgever** retourneert de **VC** aan de wallet.




**Figuur 3. Credential uitgifte via Pre-Authorized Code Flow**

Nu we de technische stromen hebben behandeld, loopt de rest van deze gids uit hoe je Keycloak kunt configureren om verifieerbare credentials uit te geven met de **Pre-Authorized Code Flow**, inclusief realm, client scope en client-level setup.

# Keycloak configureren voor OpenID4VCI

OpenID4VCI in Keycloak wordt aangeboden via de feature-vlag, die daarom bij het opstarten ingeschakeld moet worden. `oid4vc-vci`

```
--features=oid4vc-vci
```

Dat gezegd hebbende, gaan we er nu van uit dat je een Keycloak-instantie hebt met de feature-vlag ingeschakeld. `oid4vc-vci`

Naast het inschakelen van de feature-vlag, moeten de volgende configuratiestappen consequent worden uitgevoerd, in overeenstemming met uw use case:

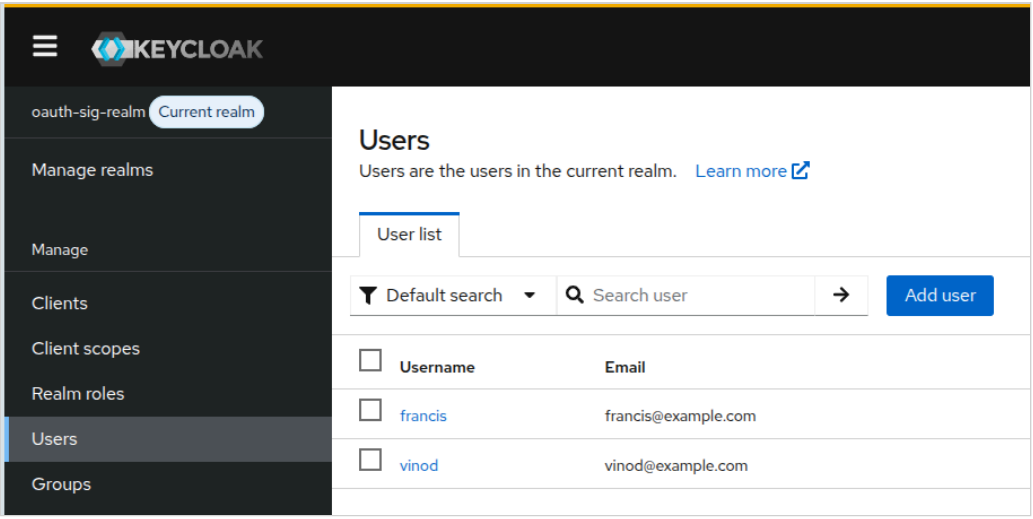
- Configureer OpenID4VCI op realm-niveau
- Configureer een uitreikbaar verifieerbaar credential (als een dedicated client scope)
- Schakel OpenID4VCI in en configureer het op clientniveau

Omdat sommige configuraties nog niet worden ondersteund via de Keycloak Admin Console, gebruiken we vaak de Admin REST API om de benodigde configuraties te communiceren.

## OpenID4VCI configureren op realm-niveau

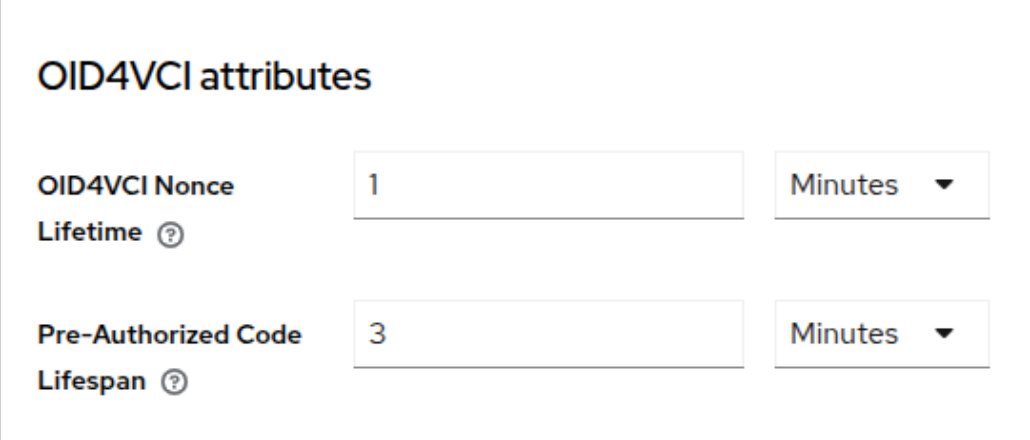
Stel dat je al een realm hebt gemaakt die de OAuth SIG-groep vertegenwoordigt, en dat er ook een paar gebruikers in deze realm zijn aangemaakt, elk met een wachtwoord toegewezen, om leden van de groep te vertegenwoordigen. `oauth-sig-realm`

Daarnaast moet elke gebruiker die credential-aanbiedingen moet aanmaken de ingebouwde rol krijgen. Deze rol wordt geleverd door de OpenID4VCI-extensie en hoeft niet handmatig te worden aangemaakt. Door deze toe te wijzen kunnen alleen geautoriseerde gebruikers credentialaanbiedingen genereren, waardoor de beveiliging van het uitgifteproces wordt gehandhaafd. `credential-offer-create`




**Figuur 4. Screenshot: Realm en Gebruikers**

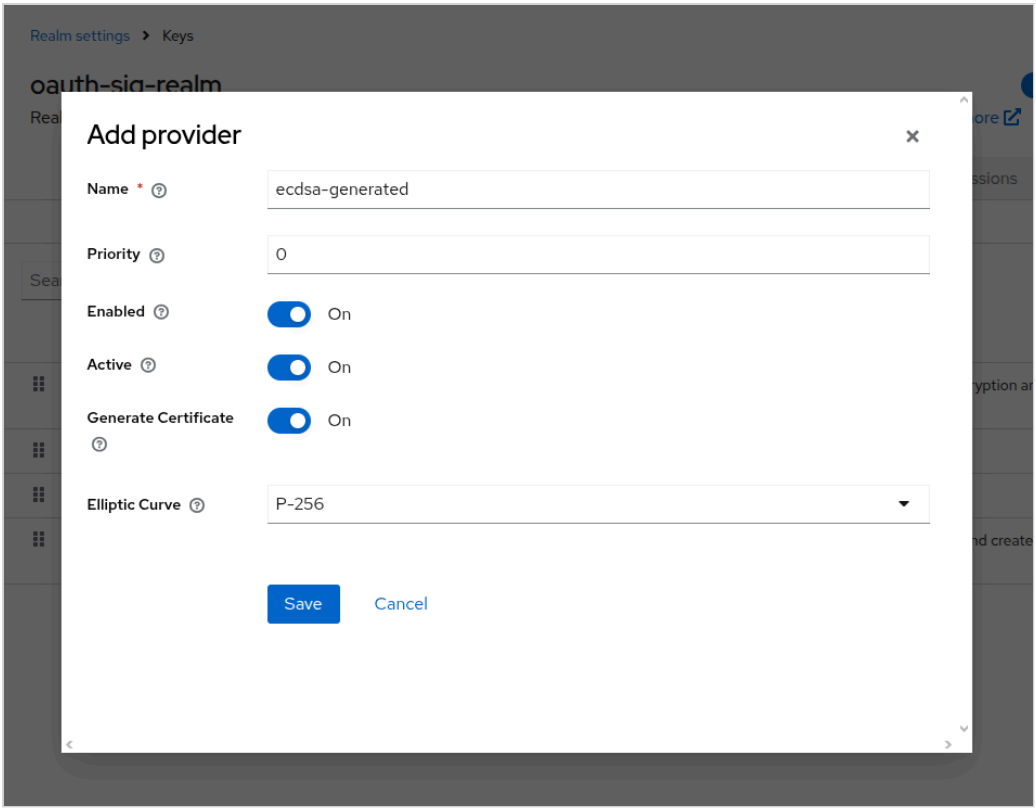
Het gedrag van OpenID4VCI over het gehele domein kan worden gemoduleerd via realm-attributen. Vind in de Admin Console het **OID4VCI Attributen-gedeelte** onder het tabblad **Realm Settings > Tokens**.




**Figuur 5. Screenshot: OID4VCI Attributen**

Verstandige standaarden gelden direct uit de doos, maar voor deze demo willen we bijvoorbeeld de **levensduur van de Pre-Authorized Code** verhogen naar 3 minuten zodat de uitgiftestroom minder snel afloopt. Voor meer informatie over de **Nonce Lifetime** en andere realm-attributen voor OpenID4VCI, raadpleeg de hoofddocumentatie van Keycloak.

Tot slot, omdat EC-cryptografie sterk wordt aangemoedigd in het OpenID4VC-ecosysteem, nodigen we u ook uit om een **P-256 Elliptic Curve-sleutelpaar** toe te voegen aan de set sleutels van het realm onder **Realm Settings > Keys > Providers**. Dit maakt later het mogelijk om het ondertekenen van uitgegeven VC's met algoritme te configureren. ES256




Figuur 6. Screenshot: ECDSA Key Provider toevoegen

# Een uitgegeven verifieerbaar credential configureren (als een dedicated client scope)

Verschillende soorten inloggegevens kunnen worden geconfigureerd voor uitgifte, afhankelijk van welke gegevens het inloggegevens moet inbedden, welk formaat het moet hebben, of andere criteria. Elk type inloggegevens is geconfigureerd als een toegewijde clientscope van protocoltype "OpenID for Verifiable Credentials".

We zullen ons lidmaatschapsbewijs zo instellen dat het wordt uitgegeven als een SD-JWT-inloggegevens met de voornaam, achternaam en e-mailadres van een gebruiker. Elke claim die aan het credential wordt toegevoegd, vereist een bijbehorende protocol mapper van het User Model naar het credential. Andere velden zoals het tijdstip van uitgifte of een unieke identificatie voor het credential worden ondersteund via andere soorten protocolmappers.

Omdat we de Admin REST API zullen gebruiken om de clientscope te configureren, moeten we eerst een geldige Admin Token uit de realm halen om onze verzoeken te autoriseren. `master`

```
ADMIN_TOKEN=$(curl -s -X POST "http://<keycloak.instance>/r
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "username=<admin-username>" \
  -d "password=<admin-password>" \
  -d "grant_type=password" \
  -d "client_id=admin-cli" | jq -r '.access_token' )
echo "Admin Token obtained: $ADMIN_TOKEN"
```

Laten we nu doorgaan met het toevoegen van het lidmaatschapsinloggegevenstype als een toegewijde clientscope met behulp van de Admin REST API.

```
curl -X POST "http://<keycloak.instance>/admin/realms/oauth
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -d '{
    "name": "membership-credential",
    "protocol": "oid4vc",
    "attributes": {
      "include.in.token.scope": "true",
      "vc.format": "dc+sd-jwt",
      "vc.verifiable_credential_type": "https://credentials
      "vc.credential_signing_alg": "ES256",
      "vc.display": "[{\"name\": \"OAuth SIG Membership\",
      "vc.credential_build_config.token_jws_type": "dc+sd-j
    },
    "protocolMappers": [
      {
        "name": "given_name-mapper",
        "protocol": "oid4vc",
        "protocolMapper": "oid4vc-user-attribute-mapper",
        "config": {
          "claim.name": "given_name",
          "userAttribute": "firstName",
          "vc.display": "[{\"name\":\"Given Name\",\"locale
        }
      },
      {
        "name": "family_name-mapper",
        "protocol": "oid4vc",
        "protocolMapper": "oid4vc-user-attribute-mapper",
        "config": {
          "claim.name": "family_name",
          "userAttribute": "lastName",
          "vc.display": "[{\"name\":\"Family Name\",\"local
        }
```

```
      },
      {
        "name": "email-mapper",
        "protocol": "oid4vc",
        "protocolMapper": "oid4vc-user-attribute-mapper",
        "config": {
          "claim.name": "email",
          "userAttribute": "email",
          "vc.display": "[{\"name\":\"Email\",\"locale\":\"
        }
      },
      {
        "name": "iat-mapper",
        "protocol": "oid4vc",
        "protocolMapper": "oid4vc-issued-at-time-claim-mapp
        "config": {
          "claim.name": "iat",
          "truncateToTimeUnit": "HOURS",
          "valueSource": "COMPUTE"
        }
      }
    ]
  }'
```

Commenting on the above configuration:

- The attribute defines the credential format (e.g. ). `vc.format dc+sd-jwt`
- The attribute defines the value of the claim inside the credential. `vc.verifiable_credential_type vct`
- The attribute specifies the cryptographic algorithm used to sign the credential. `vc.credential_signing_alg`
- All entries are intended to be used by a wallet to display intelligible descriptions. `vc.display`

To learn more about other used or available configuration attributes, please refer to the main Keycloak documentation.

Verify that the client scope was created successfully by checking the Admin Console under **Client Scopes**.

Additionally, check the Credential Issuer Metadata Endpoint by navigating to **Realm Settings**, toggling the **Verifiable Credentials** option, and saving the changes. Once done, the endpoint will be listed, allowing you to verify that the newly created credential type appears among the issuable credentials, as shown below.

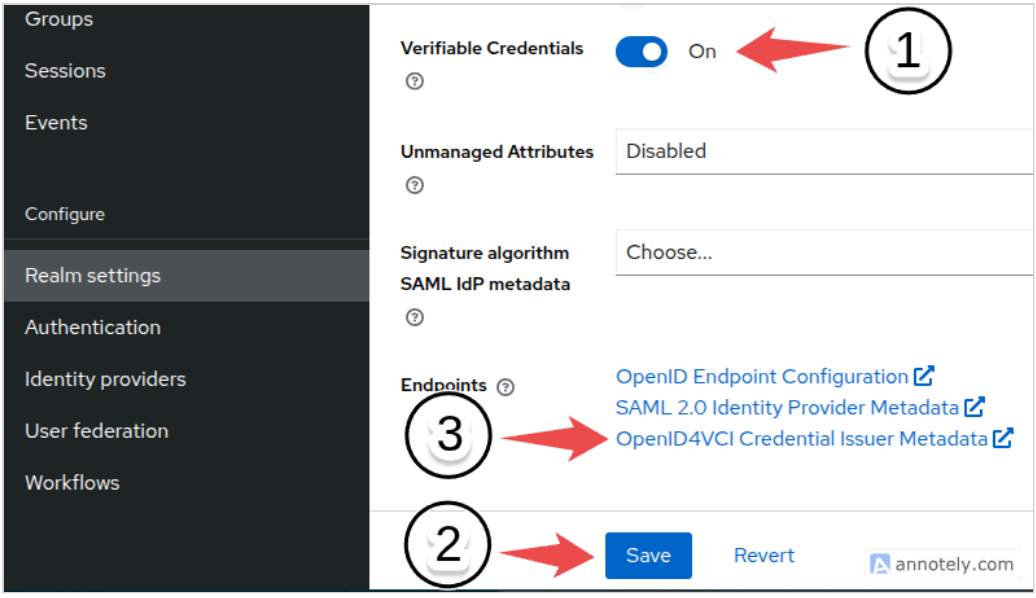



**Figure 7. Screenshot: Enable Verifiable Credentials in Realm Settings**

The credential type should appear in the metadata like the following:

```
{
  // ...
  "credential_configurations_supported": {
    "membership-credential": {
      "id": "membership-credential",
      "format": "dc+sd-jwt",
      "scope": "membership-credential",
      "cryptographic_binding_methods_supported": [
        "jwk"
      ],
      "credential_signing_alg_values_supported": [
        "ES256"
      ],
      "vct": "https://credentials.example.com/oauth-sig-mem
      // ...
    }
  }
  // ...
}
```

# Enabling OpenID4VCI at the client level

Next, let's create a new client in the realm to represent an application that members of the OAuth SIG will use to request and receive their membership credentials. We will name this client and configure it as a standard OpenID Connect client. `oauth-sig-realm` `oauth-sig-client`

During client creation, make sure to check the **Direct Access Grants** box, as we will use the **Resource Owner Password Credentials** flow to obtain a user access token in a later step. All other fields are left unchanged.

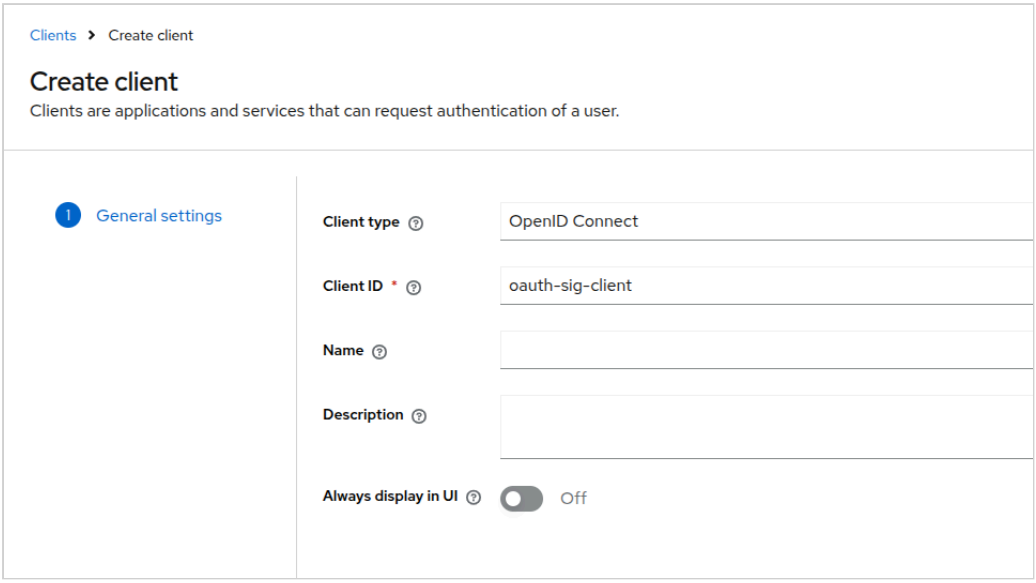



**Figure 8. Screenshot: Client Creation**

One important point to note is that clients must explicitly enable OpenID4VCI to be able to use it. Navigate to the **Advanced** tab of the client and toggle the **Enable OID4VCI** switch under the **OpenID for Verifiable Credentials** section. `oauth-sig-client`

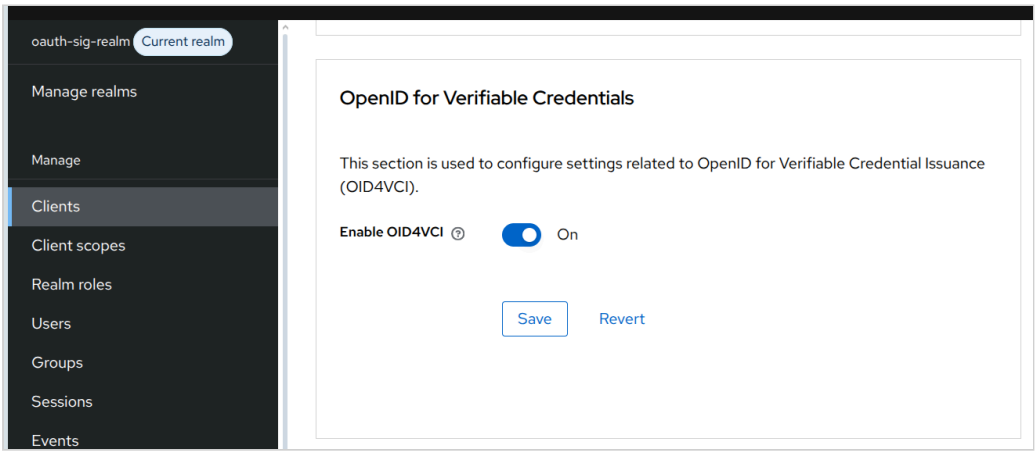



**Figure 9. Screenshot: Enable OID4VCI on Client**

Finally, we need to assign the previously created client scope to the client. To do this, navigate to the **Client Scopes** tab of the client and add as an **Optional Client Scope**. `membership-credential oauth-sig-client oauth-sig-client membership-credential`

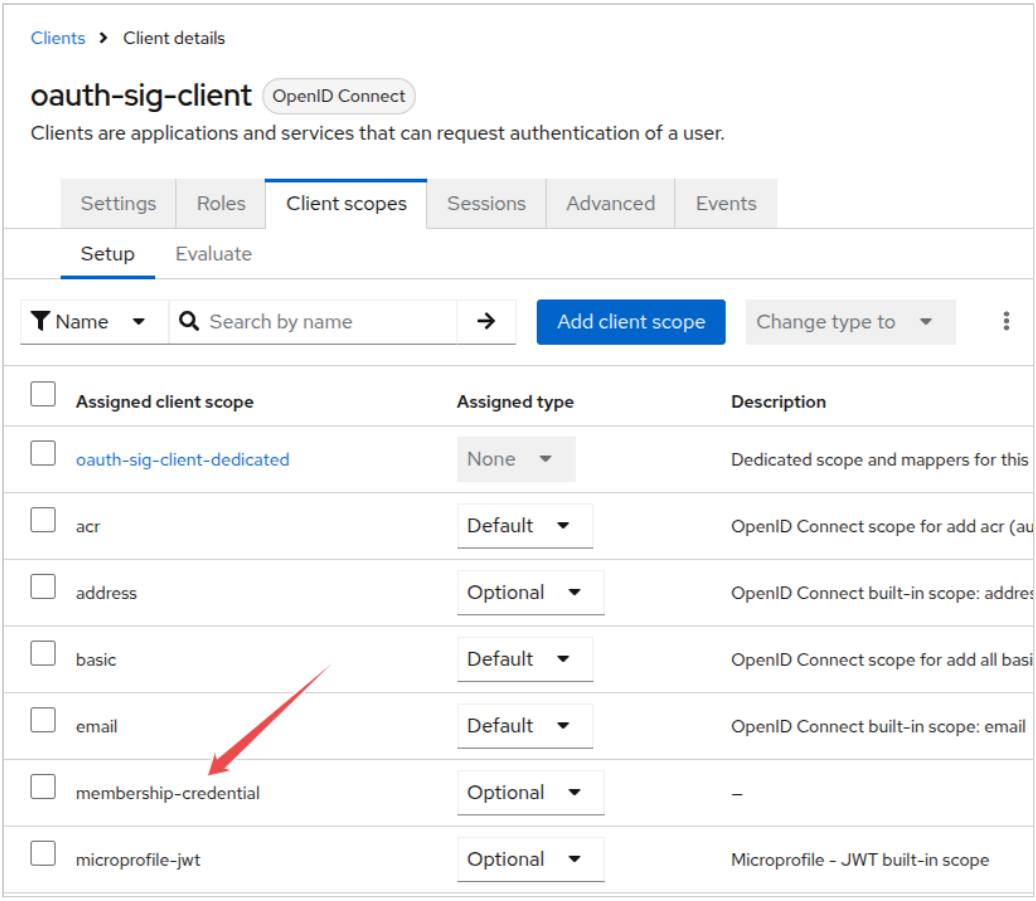



**Figure 10. Screenshot: Assigning membership-credential client scope to the client**

# Obtaining a User Access Token

Before obtaining the user access token, ensure that the user has the role, as only users with this role can create credential offers. `credential-offer-create`

With the client configured and the user's password set, we can now request a **user access token**. This token will later be used to authorize the credential offer request.

```
USER_TOKEN=$(curl -s -X POST "http://<keycloak.instance>/re
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "username=<user-username>" \
  -d "password=<user-password>" \
  -d "grant_type=password" \
  -d "client_id=oauth-sig-client" | jq -r '.access_token' )
echo "User Access Token obtained: $USER_TOKEN"
```

Now that we have a **user access token** issued to the user with the role, we can use it to request a credential offer from the issuer in the

next step. `credential-offer-create`

# Retrieving a credential offer to start the issuance flow

For Pre-Authorized Code flows, OpenID4VCI issuance can be initiated by retrieving a credential offer from Keycloak at the Credential Offer Endpoint. The endpoint requires a valid user access token and the username of the target user for whom the pre-authorized offer will be generated. In our case, we will be using the same user for both credential offer creation and retrieval.

```
curl -X GET "http://<keycloak.instance>/realms/oauth-sig-re
  -H "Authorization: Bearer $USER_TOKEN" \
  --output credential-offer-qr.png
```

Replace with the user you created earlier. The QR code image is saved as . Open this file and use your wallet to redeem the credential. `<username>` `credential-offer-qr.png`

With the query parameter , the endpoint returns a **direct binary representation of the QR code image**. If omitted, the endpoint returns JSON, which requires extra steps to construct the QR code from the offer data. `type=qr-code`

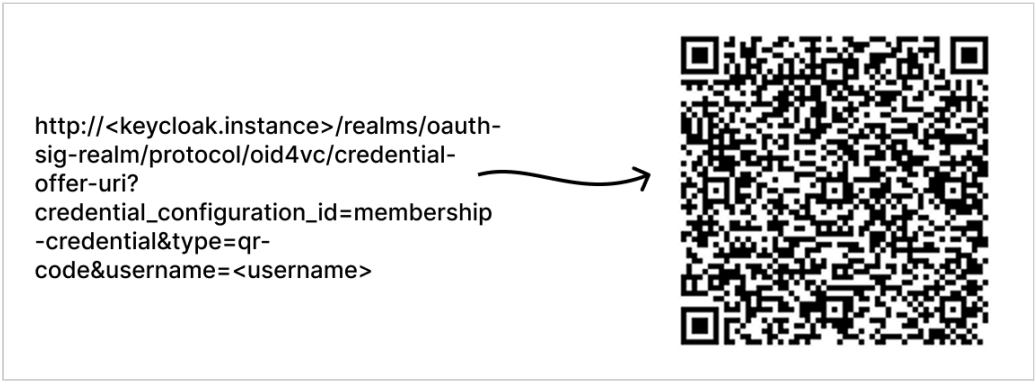



**Figure 11. QR Code Generation Overview**

Upon scanning the QR code with a compatible wallet, a membership credential is issued to the requesting user.

# ⚠️ Wallet Compatibility Notice

Keycloak implements the **final OpenID4VCI specification** and includes a limited **Draft-15 compatibility patch** with partial support.

Tested Draft-15 wallets that work with the compatibility patch:

| Wallet | Compatibility |
|---|---|
| **Heidi Wallet** | ✅ Works via Draft-15 compatibility patch |
| **Valera Wallet** | ✅ Works via Draft-15 compatibility patch |
| **Lissi Wallet** | ✅ Works via Draft-15 compatibility patch |

Full support requires wallets providers to implement the final OpenID4VCI specification.

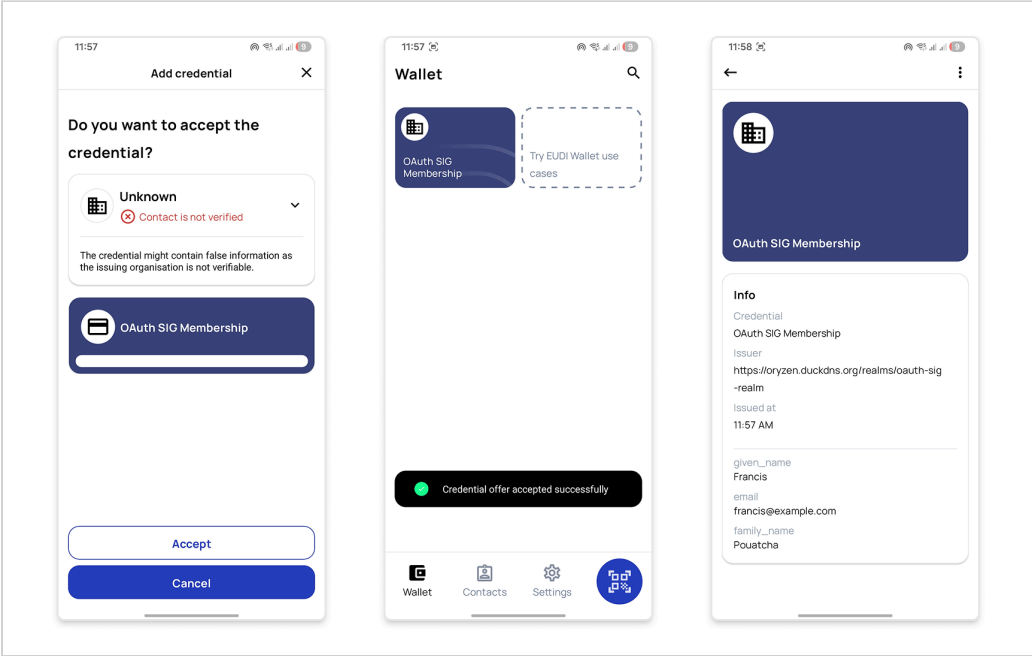



**Figure 12. Screenshot: Lissi Wallet Test**

# Before You Go

In this blog post, we have illustrated how to set up Keycloak for issuing verifiable credentials over OpenID4VCI, using a simple scenario of issuing membership credentials to members of the Keycloak's OAuth SIG group. We covered the necessary configuration steps at the realm,

client scope, and client levels, and demonstrated how to retrieve a credential offer to initiate the issuance flow.
If you are looking into streamlining this configuration process for OpenID4VCI in Keycloak, take a look at our OID4VCI Deployment project, which provides solid examples for both the pre-authorization and authorization code flows. You may also find the Keycloak Playground OID4VCI demo useful as a hands-on demo.

# Feedback & Discussion

We'd love to hear your thoughts on this guide! You can provide feedback or ask questions through:

- Slack: Join the Cloud Native Computing Foundation (CNCF) Slack and discuss with us in the channel #keycloak-oauth-sig.
- GitHub: Inspect and participate in recent OpenID4VCI-related GitHub discussions. Review existing OpenID4VCI-related GitHub issues and feel free to comment on or upvote them, or create a new issue if you have ideas for enhancements or discover a bug.

Your input will help us to improve the OpenID4VCI experience in Keycloak.

# Important Note on OpenID4VCI Development Status

OpenID4VCI support in Keycloak is still under active development. The instructions and configuration options described in this blog post are based on **Keycloak 26.5.0** and may change in future Keycloak versions as the feature evolves.

If you want to explore the latest updates to the OID4VCI feature, you can use the latest Keycloak nightly release. However, be aware that the instructions in this blog post may not work with that version.

---